香港中文大學
The Chinese University of Hong Kong

CENG3420

Lab Overview & Introduction to RARS

Fangzhou Liu

Department of Computer Science & Engineering
Chinese University of Hong Kong
{fzliu23}@cse.cuhk.edu.hk

Spring 2026

Outline

Overview of CENG3420 Labs

Assembler & Simulator

- Assembly language – symbolic
- Machine language – binary

- Assembler is a program that
  - turns symbols into machine instructions (e.g., riscv64-unknown-elf-as)

- Simulator is a program that
  - mimics the behavior of a processor
  - usually written in high-level language (e.g., spike)

CENG3420 Lab Contents

We have 3 labs in total, with 2-3 sub-labs for each lab.

- **Lab1**: RISC-V assembly language programming using **RARS** simulator.
- In lab1, we will practice coding in RISC-V assembly language, and understand how our codes run in a RISC-V CPU.
  - **Lab1-1**: basic operators and system call.
  - **Lab1-2**: function call and simple algorithm implementation.
  - **Lab1-3**: stack data structure, recursive function call, more complex algorithm implementation.

CENG3420 Lab Contents

We have 3 labs in total, with 2-3 sub-labs for each lab.

- **Lab2**: build(complete) a C-based RISC-V assembler and simulator.
- Codebase: `https://github.com/seanzw/CUHK-CENG3420-Lab`. We need to implement the assembler and simulator based on the codebase.
  - Lab2-1: implement a RISC-V assembler.
  - Lab2-2: implement a RISC-V ISA simulator with:
    - RISC-V 32 general-purpose registers
    - 32-bit data and address
    - 25+ instructions (including pseudo instructions)

CENG3420 Lab Contents

We have 3 labs in total, with 2-3 sub-labs for each lab.

- **Lab3**: build a more complete C-based RISC-V Simulator based on lab2.
  - Lab3-1: control logic in CPU, finite state machine.
  - Lab3-2: execution model, memory interface.
  - Lab3-3: BUS driver, etc.

RISC-V ISA Simulator – RARS

What is RARS

- **RARS is the RISC-V Assembler, Runtime and Simulator for RISC-V assembly language programs.**
- We write codes in RISC-V assembly language, then **RARS** translates them into RISC-V instructions and corresponding machine codes, then execute the codes through simulation, like a RISC-V CPU.
- **RARS** supports RISC-V IMFDN ISA base (riscv32 & riscv64).
- **RARS** supports debugging using breakpoints like *ebreak*.
- **RARS** supports side by side comparison from psuedo-instruction to machine code with intermediate steps.

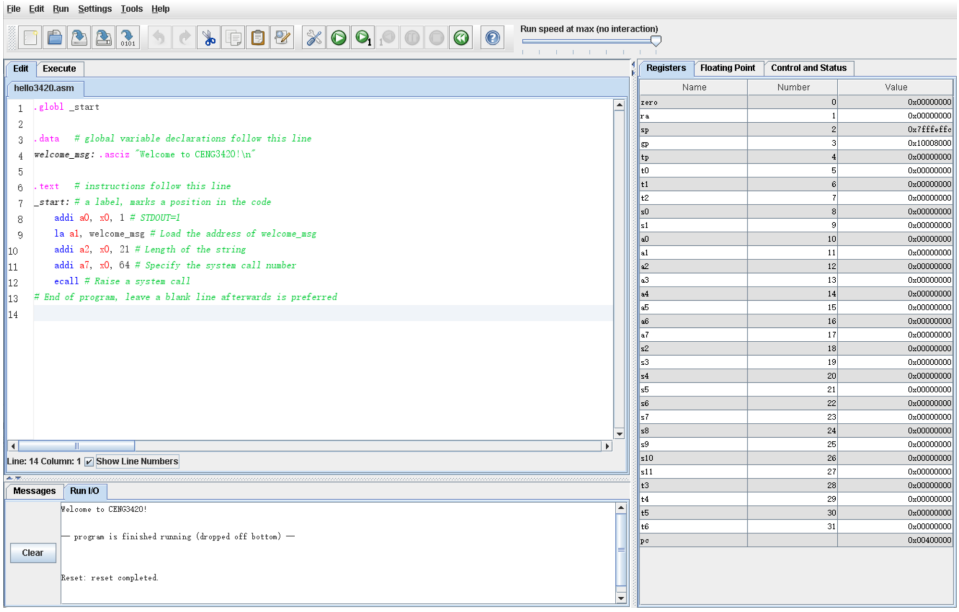
Launch RARS

- RARS tutorial: https://cass-kul.github.io/tutorials/rars/
- Install Java environment: https://java.com/en/download/
- Dowload RARS: https://github.com/TheThirdOne/rars/releases/tag/continuous
- Run RARS: run command java -jar <rars jar path> in the command window, under the path where you place rars.jar

```
cbai@hpc1:/research/dept8/gds/cbai/ta/rars$ java -jar rars.jar
```

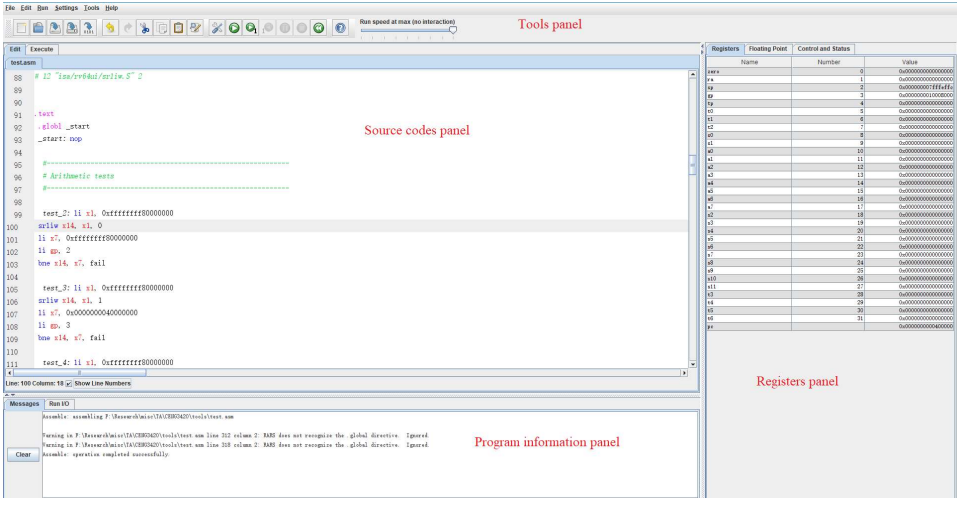
- We also provide Java install package and RARS in RARS.zip on **Blackboard**.

RARS Overview

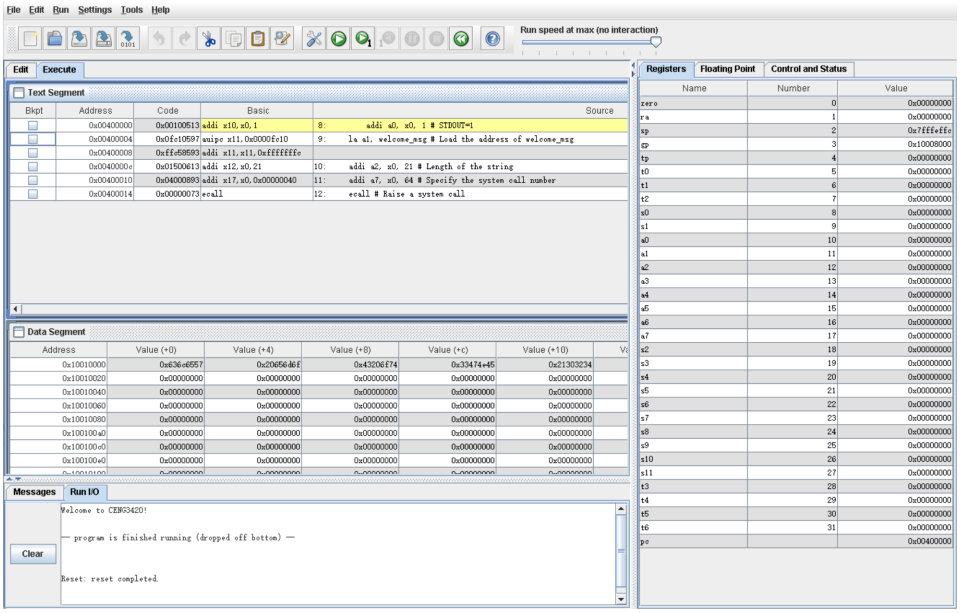


RARS edit panel

RARS Overview

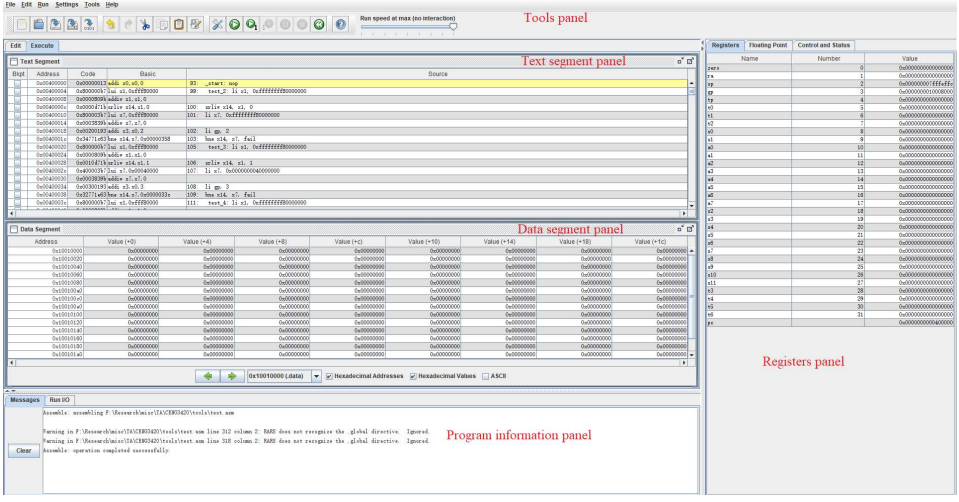


RARS Overview



RARS execute panel

RARS Overview



Shortcuts in Windows

- Create a new source file: Ctrl + N
- Close the current source file: Ctrl + W
- Assemble the source code: F3
- Execute the current source code: F5
- Step running: F7
- Instructions & System call query: F1

An Example Program

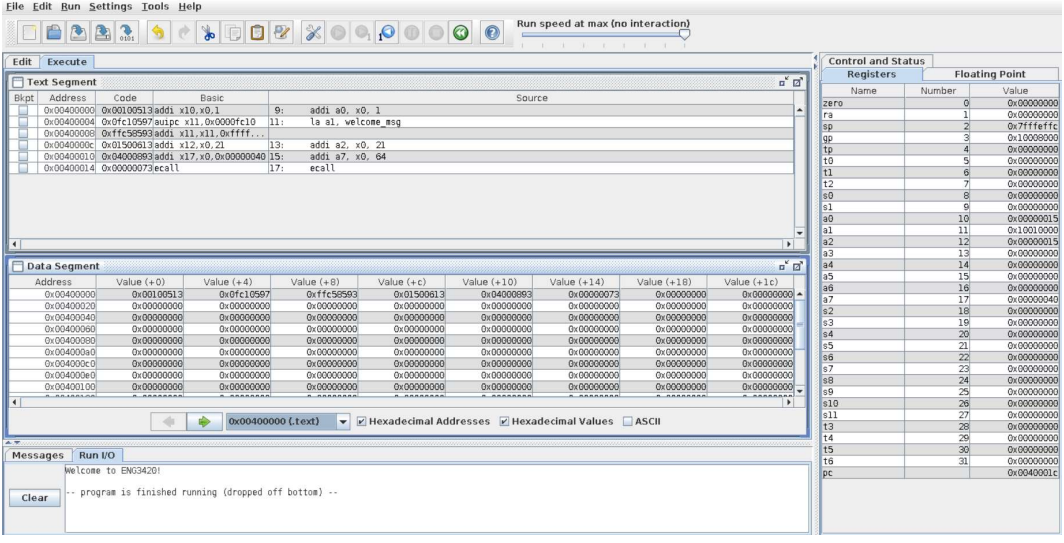
Hello CENG3420

```
.globl _start

.data   # global variable declarations follow this line
welcome_msg: .asciz "Welcome␣to␣CENG3420!\n"

.text   # instructions follow this line
_start: # a label, marks a position in the code
    addi a0, x0, 1 # STDOUT=1
    la a1, welcome_msg # Load the address of welcome_msg
    addi a2, x0, 21 # Length of the string
    addi a7, x0, 64 # Specify the system call number
    ecall # Raise a system call
# End of program, leave a blank line afterwards is preferred
```

An Example Program



System Calls in RARS

RARS provides a small set of operating system-like services through the system call (`ecall`) instruction. Register contents are not affected by a system call, except for result registers in some instructions.

- Load the service number (or number) in register a7.
- Load argument values, if any, in a0, a1, a2 ..., as specified.
- Issue `ecall` instruction.
- Retrieve return values, if any, from result registers as specified.

System Calls in RARS

Name	Number	Description	Inputs	Outputs
PrintInt	1	Prints an integer	a0 = integer to print	N/A
PrintFloat	2	Prints a float point number	fa0 = float to print	N/A
PrintString	4	Prints a null-terminated string to the console	a0 = the address of the string	N/A
ReadInt	5	Reads an int from input console	a0 = the int	N/A
ReadFloat	6	Reads a float from input console	fa0 = the float	N/A
ReadString	8	Reads a string from the console	a0 = address of input buffer, a1 = maximum number of characters to read	N/A
Open	1024	Opens a file from a path Only supported flags (a1), read-only (0), write-only (1) and write-append (9)	a0 = Null terminated string for the path, a1 = flags	a0 = the file decriptor or -1 if an error occurred
Read	63	Read from a file descriptor into a buffer	a0 = the file descriptor, a1 = address of the buffer, a2 = maximum length to read	a0 = the length read or -1 if error
Write	64	Write to a filedescriptor from a buffer	a0 = the file descriptor, a1 = the buffer address, a2 = the length to write	a0 = the number of charcters written
LSeek	62	Seek to a position in a file	a0 = the file descriptor, a1 = the offset for the base, a2 is the begining of the file (0), the current position (1), or the end of the file (2)}	a0 = the selected position from the beginning of the file or -1 is an error occurred

THANK YOU!